International Journal of General
Engineering and Technology (IJGET)
ISSN(P): 2278-9928; ISSN(E): 2278-9936
Vol. 6, Issue4, Jun – Jul 2017; 29-40

International Academy of Science,
Engineering and Technology
Connecting Researchers; Nurturing Innovations

VIRTUAL MACHINE VS CONTAINER: AN APPLICATION PERFORMANCE REVIEW

M. SRIRAGHAVENDRA[1] & PRATEEK JAIN[2]

[1]School of Computer Science and Engineering, Lovely Professional University, Phagwara, Punjab, India

[2]School of Computer Application, Lovely Professional University, Phagwara, Punjab, India

ABSTRACT

Cloud Computing is one of the main advancements that offer a promising future. One of the fundamental tools which are making the Cloud a reality is virtualization. Cloud computing provides various types of services such as Software as a service, PaaS, IaaS etc. PaaS vendors confront challenges in proficiently furnishing services with the huge growth of their offerings and hosting the distributed Applications. Hypervisor based virtualization comes with higher resource and operational overhead due to an extra level of abstraction. Virtual machine gives high service downtime when the application is updated to the new version. Containers have a favorable position over virtual machines because of performance enhancements and reduced start up time. Containers are particularly strong in managing PaaS clouds, such as application building, shipping and orchestration. Lightweight containers being the recent trends in the cloud scenario are also evolving as an important stage in PaaS service with the rise of Docker. In this paper, we have presented the performance overheads and application performance comparison among traditional hypervisor and container based virtualization.

KEYWORDS: Cloud Computing, Container, Docker, Paas, Iaas, Virtualization

NOMENCLATURE

PaaS-Platform as a service, IaaS: Infrastructure as a service, SaaS-Software as a service, VM-Virtual machine, OS-Operating system, pCPU-physical CPU, vCPU-virtual CPU.

INTRODUCTION

The advancement of technology has recently been associated with the development of problem size that we have to deal with. The problem size is expanding exponentially and physical machine can't support for a huge range of clients on various operation environments all the while. That is one of the reasons prompting to the approach of cloud computing and virtualization technology. Virtualization technology is one of the vital pieces of the cloud computing framework. Cloud computing will provide various types of services such as SaaS, PaaS, IaaS. In this paper we will concentrate only on PaaS [1]. Platform as a Service provides the abstraction required to ensure that an application independent with the Hardware infrastructure.

Virtualization technology benefits the personal computers and information technology ventures by empowering clients to share costly equipment assets by multiplexing VMs[2] (or) Containers on a single host with same Hardware configuration. The motivation behind a VM (or) Container is to upgrade resource sharing with numerous clients and enhance PC execution regarding resource usage and application flexibility. Using Virtualization strategies we can improve the utilization of CPU, Memory, networks and storage. With adequate storage, any computer platform can be installed on another host computer, regardless of the possibility that they utilize processors with various instruction sets and keep

running with particular operating systems on a similar hardware. The main benefits of virtualization include hardware independence, isolation, secure user environments, and increased scalability. Presently there is lot of virtualization technologies[3], for instance KVM, VMware Workstation, ESX, Xen, compared these virtualization technologies[4], Containers has their own great advantage[5]. Container technology is a prospering field in cloud computing. Containers implement operating system level virtualization[6], I.e. Each Host running multiple isolated Linux system Containers. Containers are lightweight than VMs. For portable[7], interoperable, distributed applications which are deployed in heterogeneous cloud environments, we require lightweight packaged Containers .The Containers deployed the applications rapidly regardless of server hardware. Containers are the best solution for more interoperable applications deployment in the cloud. Lot of the PaaS vendors prefers the Containers to reduce the Overhead of VM creation for each Application. Containers take less Startup time and service downtime than VMs. But even a hardware platform is running appropriately is no guarantee that the workloads on that platform are providing a sufficient level of service to clients. For example, one server may have the sufficient resources to run the Applications, yet workloads may still face conflicts among Applications, mainly with shared resources. So the significance of every application focusing is gradually moving from System management to Application Service management. Application management intended to guarantee that workloads are delivering an appropriate level of performance to end users. Containerization provides the Automated Tools for Application packaging, Delivering and orchestrating services and applications.

RELATED WORK

Xavier et al. [10] evaluated the performance of Linux-Server, OpenVZ, LXC for hpc, using the NPB Benchmark suite. It evaluates the performance evaluation and isolation with HPC workloads. In their experiment they found that all Container-based systems have a near native performance, but the differences between the Container technologies lies in the resource management implementation. Container based technologies provide poor performance isolation for Memory, Disk and Network except the CPU.

Liu and Zhao [9] describes about the importance and architecture of the Docker. This paper explains how the Docker play the role in rapid Application development, portability across different execution environment.

Dua et al. [10] compared the VMs with Containers on multiple factors. They discussed about the PaaS Use case and Container based PaaS architectures with existing vendors. They explore the various container implementations with some specific parameters.

Scheepers [11] presented the performance comparison between Xen and LXC Container, he also discussed about operational flexibility of technologies. A macro benchmark has been performed on Application performance and inter virtual machine communication.

Gerlach et al. [12] did a comprehensive study where he implement a Docker Container technology in Skyport Scientific workflows. They specifies the limitations of the existing workflow platforms and implementing the Docker Linux Containers in AWE/Shock data Analysis platform for development of software Applications with their own execution environment.

Morabito et al. [13] compared Native, Docker, KVM, OSv with selected Benchmark Tools to measure CPU, Memory, Disk I/O, Network I/O performances. In this paper Y-cruncher, NBENCH, Linpack Benchmark Tools used for CPU performance, Bonnie++ for Disk I/O, STREAM for Memory performance, Netperf for Network I/O performance.

The results shown that, Containers are achieved better performance when compared with virtual machines.

Barik et al. [14] investigated in terms of performance analysis for VMs and Containers with some Benchmark Tools. The Tools which are used in this paper are, AIO stress (i.e. write operation in the local and global Thread memories),Ram-speed (memory access operations on level 1 & 2 cache and main memory), IO Zone (random read/write operations on the system file (or) Disk),Network loop-back (TCP,UDP performance among networks), Build Apache (the time to build, compile and run an Apache HTTP Web server), Build PHP (the time to build the php5 server along with zend engine), Apache (how many requests per second a system being carried out concurrently), G-crypt (how much time it takes to do the encryption and decryption with 100 repetitions), Blake 2 (measure the time taken to cipher and decipher the input), 7-Zip(MIPS value),Open SSL (number of Open SSL certificates it has signed in the web).These Tests concluded that Containers perform well over VMs except security, cryptographic tests.

Maliva et al. [15] They proposed a multi-task cloud infrastructure using Docker and AWS services for rapid deployment, application optimization and Isolation. They conclude that we can build any app in any language, dockerized the app then app can run any ware.

Mazaheri et al. [16] They used Cloud Tester Benchmark Suite(CTBS) to study and understand the performance of the systems in three different execution environments, bare-metal, Docker and KVM. Using HPCC and IOR benchmarks in CTBS, they found that Docker delivered near bare-metal performance while KVM performance was relatively less.

Augusto Neto[17] He performed synthetic benchmark of two different virtualization technologies on a Cubieboard2 SoC platform, which represents the architecture of IoT gateways hardware. He conclude that Linux containers seems to take advantage over hypervisor based virtualization for deploying applications at the network edge.

Minh Thanh Chung et al. [18] They evaluated the performance of hypervisor based, Container based for HPC distributed Applications, using HPL Benchmark and Graph500.The evaluation mainly focuses on two criteria's consisting of computational ability and data traceability. This paper concludes that Docker is more suitable than VMs about data intensive Applications.

Lei Xu et al. [19] presented the Condroid, a light weight Android virtualization based on Container technology. Condroid uses the single kernel for several Android containers. It does the resource isolation, resource control based on namespace and cgroup. Condroid reduce the memory and storage utilization using service sharing, file system sharing mechanism. In this paper they demonstrate the Nexus 5 running with Condroid and conclude that Condroid gives near zero performance overhead than Cells.

Shuangshuang Shen et al. [20] proposed the new architecture of SaaS system server environment with container based virtualization for migration and deployment. Container based virtualization provides the "semi frozen" mechanism for rapid automatic migration, It saves the cost for SaaS providers.

Claus Pahl and Brian Lee [21] are using the container based technology in application packaging and orchestrating the deployed services for edge cloud environment. They describe the cluster topology orchestration adopted from the TOSCA standard.

VIRTUALIZATION TECHNOLOGY OVERVIEW

Hypervisor-Based Virtualization

In this type of virtualization, mainly it isolates the hardware from the software to yield better system efficiency. It abstracts the computer hardware, system resources (CPU, memory, network, storage) and think as a single machine to go even it has many Virtual machines.Each Host machine can run multiple VMs with the Hypervisor based middleware software (VMM). Each Virtual Machine consists full Guest OS in addition to the Binaries and Libraries necessary for the Applications. The Hypervisor lies between the Hardware and Guest OS.

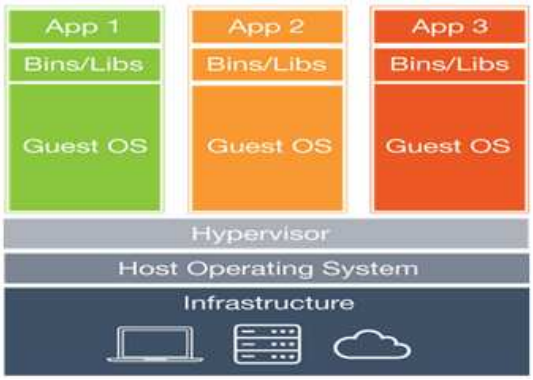

Figure 1: Virtualization Architecture

There are different Hypervisors in the market, each with their unique advantages and disadvantages. The Hypervisors broadly classified into two types:

- In Type 1, the Hypervisor software, insert directly on top of the hardware. We can also call the Native virtual machine. It provides Hardware virtualization. Type 1 provides more performance than Type 2. Ex: Xen, VMware ESXi.
- In Type 2, the Hypervisor software runs on the top of Operating system (Host OS). We can also call the Host based virtual machine. It provides the Software virtualization. Type 2 allows multiple versions of the Hypervisors in single Host machine, with that it provides more flexibility than Type 1. Ex: QEMU, Virtual Box.

  There are mainly three different virtualization techniques:

Full Virtualization

It does not modify the Guest OS. The Guest OS is not aware about the virtual environment, because of Guest OS is completely decoupled with the Hardware. The Hardware is virtualized by the Host OS. The Guest OS executes the non privileged instructions and Traps the privileged instructions threw Binary calls to the Hypervisor software for emulation. Performance Overheads occur with the Binary translation.

Para Virtualization

It modifies the Guest OS. The Guest OS is aware of the virtual environment. It executes the non privileged instructions directly by Guest OS, when privileged instructions occur, modifications should be done in Guest OS. It reduces the Overhead, but it creates the complexity with modification of Guest OS. Hypervisor software provides the API's to Guest OS for executing the privileged instructions.

Hardware-Assisted Virtualization

This virtualization is introduced by AMD and Intel. It does not modify the Guest OS like full virtualization and also no Overhead of hardware emulation. The hypervisor software load at Ring -1, Guest OS accesses the CPU directly at Ring-0 just like when using in Host machine. The main limitation is, if more VM exits, then the performance of this virtualization should be degrading. VM exits will be caused by the I/O operations of Guest OS's.

Container-Based Virtualization

Container-based virtualization shares the single kernel with multiple Containers installed on top of it. Unlike the hypervisor-based virtualization, Container does not maintain a total operating system instance, so that the disk images of the Container are smaller than VM. Containers are isolated with each other. Container virtualization done at the OS level (OS functions) rather than Hardware level. It does not virtualizes the Hardware resources. The Host kernel performs resource management for Containers.

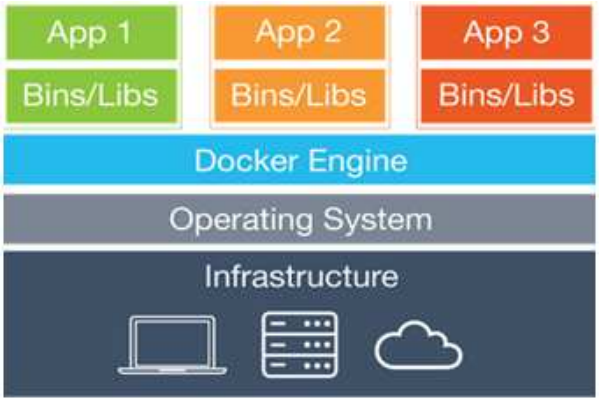

Figure 2: Container Architecture

Container-based virtualization disposes of the need to copy certain OS functions. For instance, it doesn't copy hardware calls, and one operating system is in charge of dealing with all hardware access. The main advantage of Container-based virtualization is providing Scalability and operational flexibility. Hypervisor virtualization generally has restricted on the amount of CPU and memory resources can be allocated to a Guest OS, the Container based virtualization addresses the availability of CPUs and RAM for the host kernel. Container architecture does not maintain the system Hypervisor like VMs; it's having their own Container engine with light weight. Container engine acts as the master and multiple containers are clients.

There are two types of Containers.

- Application Container
- System Container

The container which having a single application is called "**Application Container**", those who maintained the complete operating system along with multiple process and services instead of single Application is called "**System Container**".

This list below gives the details about popular Container implements

- LXC
- Docker
- OpenVZ
- Warden

With the Copy-on-Write method, every Container appeared to be having single unified image, but Multiple Containers in a machine share a common image layer for reducing the data storage. Copy-on-Write mechanism allows Container processes to share the pages rather than maintain individual copies of the pages. However, when any container process tried to write on the shared copy, then the pages should be stored in a private copy of the Container process.

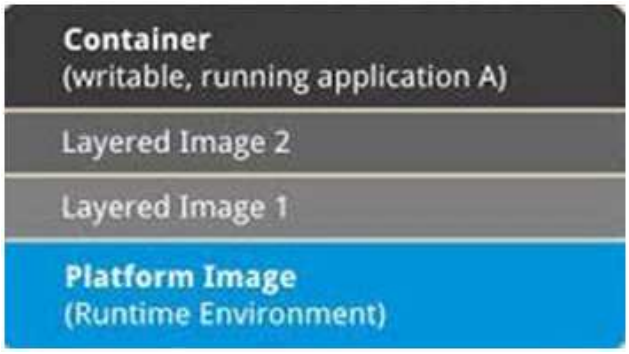

Figure 3: Docker Image

A Container is created from the Docker image. In the above diagram the first layer from the bottom is the Base image or Platform Image lies on Kernel. The Base image includes either debain, ubuntu, busybox, fedora or cent os. Second and third layer are an Image1 and Image2, these are the read-only layers. We can maintain any number of Image layers on Base image. The top layer is the read-write layer i.e. Container. The union file system creates this read-write layer to update into the new image.

The Kernel encourages the execution of Containers by using namespaces for resource isolation, and cgroup for resource administration. Every Container has their own independent file system and network stack.

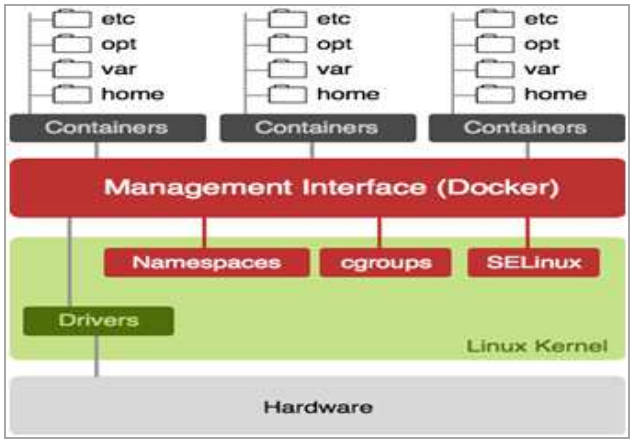

Figure 4: Namespace and Cgroup

Containers can be intricate, difficult to set up, and hard to manage and automate. Docker plans to change that. Docker is a new Container technology, it is an open-source engine. Docker provides to Build, Ship and Run the distributed application anywhere. The Docker Engine Container includes only the application and its dependencies. Developers, System Admins and Enterprise get benefited with this Docker technology. With the help of libcontainer library the Docker will create the Containers with complete isolate form.

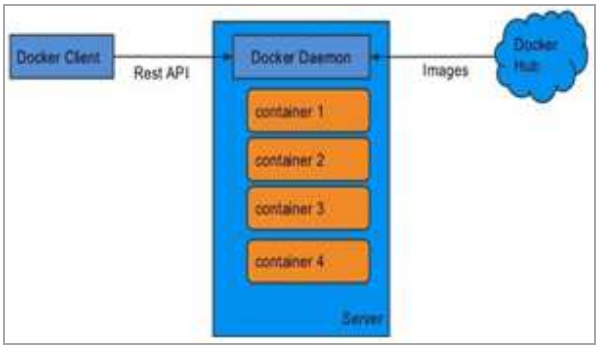

Figure 5: Docker Architecture

Docker is a client-server model. Docker client communicates with the Docker server threw the API. Docker server is called as a daemon or Docker Engine. Docker Engine executes the Containers within the Host. Docker client and Docker server run on the same Host or Docker client communicate with the remote Docker Engine in another Host. Docker Hub is the registry to store the Images.

Here are a few limitations with respect to the Container-based virtualization. There is no direct installation of the Guest OS in Containers like Virtual machines. For running the Containers, first create the Container template or use already created templates. We need the additional requirement for saving the configuration information permanently. Containers are less flexible i.e. It supports only same operating system or similar Guest OS's, so the Linux Containers cannot allow booting on top of a Windows Host and vice versa. Containers have less secure than hypervisor virtualization because of weaker isolation compared to hypervisors. In Container-based virtualization, if stress applied to one Container affected the performance of the application running on other Container, due to the lack of isolation. We can't isolate the kernel updates in Linux Containers, updates to the host kernel impacts all Containers.

PERFORMANCE OVERHEADS OF VIRTUALIZATION

Virtualization technology provides the resource virtualization, i.e. sharing the resources between the applications. The applications on the same machine can run in different execution environments. Resource isolation needs for virtualized the system resources. To abstract the system resources, hypervisor based virtualization performs some redundant functionalities. This redundancy imposes overhead when providing the needed resource isolation[22].This paper explains the list of Overheads occurring in the VMs.

- CPU Overhead
- Memory Overhead
- Network Overhead
- Disk Overhead

CPU Overhead

- Overhead occurs with the Double Scheduling, where the hypervisor (VMM) and Guest OS will both do the CPU Scheduling. The hypervisor Schedules virtual CPUs (vCPU) on physical CPU, Guest OS Schedule processes on vCPUs. Hypervisor schedulers are unaware of Guest OS scheduling, so the performance degradation will occurs with parallel applications running in VMs[23].
- Two fundamental issues with current hypervisor schedulers are vCPU preemption and vCPU stacking[24].If the vCPU is preempted in the middle of the critical section, it holds the lock for the long time which is a significant performance degradation. The problem is also called as lock holder preemption[25]. Synchronization latency occurs when lock waiter vCPU to be scheduled before lock holder vCPU on the same pCPU is called vCPU stacking.
- Virtualization platform fails to enforce fairness for SMP virtual machine which having more than one virtual CPUs(vCPU)[26].Hypervisor scheduling assigns pCPU time equally to all Vms in the machine, even the Vms have varying vCPUs. Hypervisors avoid Vms when the usage limit of pCPU exceeds, even the pCPU is idle.
- Significant delay in interrupt handling, because the interrupt handler won't be executed until the hypervisor schedules the vCPU in a virtual machine. High CPU Overhead with interrupt handling. When an interrupt occurs the context switching done between the Guest OS and Host OS. For every disk IO operations and packet sending and receiving the interrupt will generate[27].

Memory Overhead

- Hypervisor based virtualization causes performance overheads on memory because of Memory Recovery and Memory Duplication.
- Overhead occurred with "Double paging problem" in Memory Recovery, because the Host and Guest OSs both are maintain the Memory Recovery policy. When physical memory space is overcommitted, the hypervisor needs recover memory from the virtual machines it managed and expel pages from physical memory. Hypervisor does not have the sufficient information about Guest dirty pages to expel because the memory pages are managed by the Guest OS. There is a solution to this using balloon process[28],in which the Hypervisor expel only the memory pages which are evicted by the Guest OS.
- Every VM will maintain separate memory page table, so it occupies lot of physical memory. The VMs are isolated from each other in a machine. Each VM has individual memory copy of operating system and system libraries even though they are running the same application.

Network Overhead

- Hypervisor based virtualization causes performance overheads on Network because of Packet processing, CPU utilization, Network address Translation
- One of the overhead of Network virtualization is I/O virtualization due to the virtual interfaces. Most of the time will consumed for moving the packets from kernel and Applications. Sending the packets having more delay than receiving packets are caused by VM Scheduling.

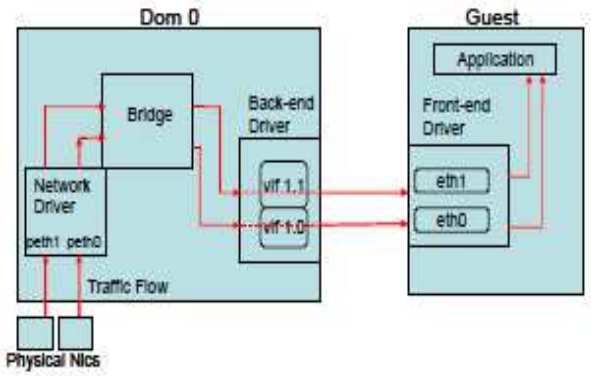

Figure 6: XennetworkI/O Processing[29]

In the above Xen architecture, Dom0 creates the virtual interface to every device in the Guest and these are activated with bridge. The packet arrives from the physical node to the physical device(petho,peth1..),it is routed through the Network Driver to bridge and to the virtual interface(vif 1.0,vif1.1,...).The Back-end Driver transfers the packets to the Front-end Driver of the Guest VM and it takes by the Application[29].Every I/O access from the Guest VM comes thru the hypervisor from Dom0.

- Network performance degradation and Unstable networks are causes with interferences among process computation and network communication for CPU utilization. Network communication should be waiting for a long time to schedule on a CPU. we recognize the two typical situations that prompt such a circumstance, that is "Co-runner interference" and "Self interference". netback process of a VM is starved because of tight scheduling with other VMs is called Co-runner interference. Within a specific virtual machine, the netfront process is starved because of tight scheduling with other processes is called Self interference. The solution for this overheads are keeping separate CPU pools for computation and communication tasks.
- In Docker all the containers are connected to bridge and the bridge is connected to network through the NAT. Network address translation gives performance overhead due to consume more CPU cycles which increases the packet delay with the more number of packets in the network[30].

Disk Overhead

- virtualization causes performance overheads on disk because of I/O scheduling and layered file systems.
- In virtualization environment Multiple VMs having their own operating systems and virtual disks are running on VMM layer. Each VM are scheduled disk I/O operations individually before passing to VMM, again the VMM will do the disk I/O operations among all VMs. There is a possibility to get the conflictions among the different scheduling algorithms between the layers, It significantly impacts on Application performance[31].
- Docker uses the layered file system, it supports the Another Union File System (AUFS). AUFS introduces significant overheads because I/O operations of file systems go through the several layers.

APPLICATION PERFORMANCE

The list of difficulties will be faced at the Build and Test phases of the project, i.e. Configuration setup, Version conflicts, dependencies missing, old data. It consumes more time to prepare Developer environment and production

environment with manual setup. There is a chance to make the mistakes with manual setup due to wrong versions of compilers, Build tools, incompatible libraries, some missing packages. It's difficult to maintain multiple versions of the same software tools in one machine. Missing the test dependencies for integration, such as API's, message brokers, databases. The possibility of using the old data on build and test machines, such as old libraries, old data from previous builds.

We can easily avoid these problems using the Docker. It helps to the Developers and System Admins as shown below.

- It is useful in local development, mainly for repeatable builds, i.e. same environment should be reproducible on the developer site and production server.
- All the test dependencies are neatly packaged and build as Docker image. Anywhere with the Docker can run that Application irrespective of environment.
- Docker can run the Application which is having multiple programming language components in a single machine.
- It supports multiple versions of the same software tool running in a one machine.
- Docker has the built in Orchestration on containers. It is balancing the load and scaling the application services according to demand.

CONCLUSIONS

The Container technology in PaaS is becoming mainstream and the Docker Containers are becoming into an accepted standard. Container reduces the Overheads on server deployment in the Cloud. It efficiently uses the Datacenter resources and provides the Application Scalability. Container technology advances the PaaS towards distributed heterogeneous multi clouds through light weight and interoperability. Both Virtual machine and cloud Container have experienced a major jump in their technological progression since their creation. Virtual machine faces the challenges with the performance Overheads, it impacts the Application performance. In this work we describe about the Virtualization and Application development Overheads.

REFERENCES

1. O. Gass, H. Meth, and A. Maedche, “Paas characteristics for productive software development: An evaluation framework,” *IEEE Internet Comput.*, vol. 18, no. 1, pp. 56–64, 2014.doi:10.1109/MIC.2014.12
2. J. E. Smith and Ravi Nair, “The architecture of virtual machines Smith, J.E. Ravi Nair,” *IEEE Comput.*, vol. 38, no. 5, pp. 32–38, 2005. doi:10.1109/MC.2005.173
3. N. Regola and J.-C. Ducom, “Recommendations for Virtualization Technologies in High Performance Computing,” *Cloud Comput. Technol. Sci. CloudCom 2010 IEEE Second Int. Conf.*, pp. 409–416, 2010.doi:10.1109/CloudCom.2010.71
4. A. J. Younge, R. Henschel, J. T. Brown, G. Von Laszewski, J. Qiu, and G. C. Fox, “Analysis of Virtualization Technologies for High Performance Computing Environments,” *2011 IEEE 4th Int. Conf. Cloud Comput.*, pp. 9–16, 2011.doi:10.1109/CLOUD.2011.29

5. M. Joy, “Performance comparison between Linux containers and virtual machines,” *Conf. Proceeding - 2015 Int. Conf. Adv. Comput. Eng. Appl. ICACEA 2015*, no. Lxc, pp. 342–346, 2015.doi:10.1109/ICACEA.2015.7164727
6. S. Soltesz, H. Pötzl, M. E. Fiuczynski, A. Bavier, and L. Peterson, “Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors,” *ACM SIGOPS Oper. Syst. Rev.*, vol. 41, no. 3, p. 275, 2007.doi:10.1145/1272998.1273025
7. S. Qanbari, F. Li, and S. Dustdar, “Toward a portable cloud manufacturing services,” *IEEE Internet Comput.*, vol. 18, no. 6, pp. 77–80, 2014.doi: 10.1109/MIC.2014.125
8. M. G. Xavier, M. V Neves, F. D. Rossi, T. C. Ferreto, T. Lange, and C. A. F. De Rose, “Performance Evaluation of Container-based Virtualization for High Performance Computing Environments,” 2013.doi:10.1109/PDP.2013.41
9. D. I. Liu and L. Zhao, “THE RESEARCH AND IMPLEMENTATION OF CLOUD COMPUTING PLATFORM BASED ON DOCKER,” pp. 475–478. doi: 10.1109/ICCWAMTIP.2014.7073453
10. R. Dua, A. R. Raja, and D. Kakadia, “Virtualization vs containerization to support PaaS,” *Proc. - 2014 IEEE Int. Conf. Cloud Eng. IC2E 2014*, pp. 610–614, 2014.doi:10.1109/IC2E.2014.41
11. M. J. Scheepers, “Virtualization and Containerization of Application Infrastructure: A Comparison,” *21st Twente Student Conf. IT*, pp. 1–7, 2014
12. W. Gerlach *et al.*, “Skyport - Container-based execution environment management for multi-cloud scientific workflows,” *Proc. DataCloud 2014 5th Int. Work. Data Intensive Comput. Clouds - Held Conjunction with SC 2014 Int. Conf. High Perform. Comput. Networking, Storage Anal.*, pp. 25–32, 2014.doi:10.1109/DataCloud.2014.6
13. R. Morabito, J. Kjällman, and M. Komu, “Hypervisors vs. lightweight virtualization: A performance comparison,” *Proc. - 2015 IEEE Int. Conf. Cloud Eng. IC2E 2015*, pp. 386–393, 2015.doi:10.1109/IC2E.2015.74
14. R. K. Barik and R. K. Lenka, “Performance Analysis of Virtual Machines and Containers in Cloud Computing,” pp. 1204–1210, 2016.doi:10.1109/CCAA.2016.7813925
15. G. M. Tihfon, J. Kim, and K. J. Kim, “Information Science and Applications (ICISA) 2016,” vol. 376, pp. 1339–1349, 2016.doi:10.1007/978-981-10-0557-2_126
16. S. Mazaheri, Y. Chen, E. Hojati and A. Sill, "Cloud benchmarking in bare-metal, virtualized, and containerized execution environments," 2016 4th International Conference on Cloud Computing and Intelligence Systems (CCIS), Beijing, 2016, pp. 371-376. doi: 10.1109/CCIS.2016.7790286
17. A. Neto, “Virtualization at the Network Edge: A Performance Comparison.”. doi: 10.1109/WoWMoM.2016.7523584
18. M. T. Chung, N. Quang-Hung, M. -T. Nguyen, and N. Thoai, “Using Docker in high performance computing applications,” *2016 IEEE Sixth Int. Conf. Commun. Electron.*, pp. 52–57, 2016.doi:10.1109/CCE.2016.7562612

19. L. Xu, G. Li, C. Li, W. Sun, W. Chen, and Z. Wang, "Condroid: A Container-Based Virtualization Solution Adapted for Android Devices," *2015 3rd IEEE Int. Conf. Mob. Cloud Comput. Serv. Eng.*, pp. 81–88, 2015.doi:10.1109/MobileCloud.2015.9

20. S. Shen, H. Huang, and X. Lin, "A new architecture of server environment of SaaS system for easier migration and deployment," *Proc. 2015 4th Int. Conf. Comput. Sci. Netw. Technol. ICCSNT 2015*, no. Iccsnt, pp. 381–386, 2016.doi:10.1109/ICCSNT.2015.7490774

21. Pahl and B. Lee, "Containers and Clusters for Edge Cloud Architectures – a Technology Review," 2015.doi:10.1109/FiCloud.2015.35

22. A. Sampathkumar, "Virtualizing Intelligent River ®: A Comparative Study of Alternative Virtualization Technologies," 2013

23. X. Song, J. Shi, H. Chen, and B. Zang, "Schedule Processes, not VCPUs," *APSys'13*, pp. 1–7, 2013.doi:10.1145/2500727.2500736

24. P. Enberg, "A Performance Evaluation of Hypervisor, Unikernel, and Container Network I/O Virtualization," 2016

25. V. Uhlig, J. LeVasseur, E. Skoglund, and U. Dannowski, "Towards Scalable Multiprocessor Virtual Machines," *Vm'04*, pp. 43 – 56, 2004

26. J. Rao and X. Zhou, "Towards Fair and Efficient SMP Virtual Machine Scheduling," *PPoPP'14*, pp. 273–286, 2014.doi:10.1145/2692916.2555246

27. "VIRTUAL INTERRUPT HANDLING TO REDUCE CPU OVERHEAD IN I / O VIRTUALIZATION Thesis Submitted in partial fulfilment of the requirements of BITS C421T Thesis By Nomchin Banga ID No . 2009B4A7751P Under the supervision of Prof . Rahul Banerjee (Chief, Softwa," no. 2009, 2013

28. a. Waldspurger, "Memory resource management in VMware ESX server," *ACM SIGOPS Oper. Syst. Rev.*, vol. 36, no. SI, p. 181, 2002.doi:10.1145/844128.844146

29. P. Apparao, S. Makineni, and D. Newell, "Characterization of network processing overheads in Xen," *VTDC 2006 - 2nd Int. Work. Virtualization Technol. Distrib. Comput. held Conjunction with SC06*, no. Vtdc, 2006.doi:10.1109/VTDC.2006.3

30. W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, "An Updated Performance Comparison of Virtual Machines and Linux Containers," *Technology*, vol. 25482, pp. 171–172, 2014.doi:10.1109/ISPASS.2015.7095802

31. Boutcher and A. Chandra, "Does virtualization make disk scheduling passé?," *ACM SIGOPS Oper. Syst. Rev.*, vol. 44, no. 1, pp. 20–24, 2010.doi:10.1145/1740390.1740396